RECEIVED
CENTRAL FAX CENTER
JUL 28 2006

PATTERSON & SHERIDAN, LLP
ATTORNEYS AT LAW

3040 Post Oak Blvd, Suite 1500
Houston, TX 77056-6582
TEL 713.623.4844
FAX 713.623.4846

FACSIMILE COVER SHEET

DATE:	July 28, 2006
FILE NO:	ROC920010193US4 (IBMK10196)
TO:	MAIL STOP APPEAL BRIEF - PATENTS
	Examiner LeChi Truong
FAX NO:	1-571-273-8300
FROM:	Gero G. McClellan/Jon K. Stewart
PAGE(S) with cover:	25

RE:

TITLE: METHOD FOR CONTINUOUS I/O REQUEST PROCESSING IN AN ASYNCHRONOUS ENVIRONMENT

U.S. SERIAL NO.:	10/037,553
FILING DATE:	January 4, 2002
INVENTOR(S):	Bauman et al.
EXAMINER:	LeChi Truong
GROUP ART UNIT:	2194
CONFIRMATION NO.:	7117

Attached are the following document(s) for the above-referenced application:

Response to Notice of Non-Compliant Appeal Brief.

PATENT
Atty. Dkt. No. ROC920010193US4
PS Ref. No.: IBMK10196

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:
Bauman et al.

Serial No.: 10/037,553

Filed: January 4, 2002

For: METHOD FOR CONTINUOUS I/O REQUEST PROCESSING IN AN ASYNCHRONOUS ENVIRONMENT

Confirmation No.: 7117

Group Art Unit: 2194

Examiner: LeChi Truong

MAIL STOP APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop Appeal Brief - Patents, Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313-1450, or facsimile transmitted to the U.S. Patent and Trademark Office to fax number 571-273-8300 to the attention of Examiner LeChi Truong, on the date shown below:

July 28, 2006
Date

Jon Stewart

Dear Sir:

RESPONSE TO NOTICE OF NON-COMPLIANT APPEAL BRIEF

In response to the Notice of Non-Compliant Appeal Brief, Appellants submit this Substitute Appeal Brief to the Board of Patent Appeals and Interferences in response to the Notification of Non-Compliant Appeal Brief dated July 19, 2006 in the Appeal of the above-identified application.

TABLE OF CONTENTS

Real Party in Interest

The present application has been assigned to International Business Machines Corporation, Armonk, New York.

Related Appeals and Interferences

Applicant asserts that no other appeals or interferences are known to the Applicant, the Applicant's legal representative, or assignee which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

Status of Claims

Claims 1, 3-10, 12-19 and 21-26 are pending in the application. Claims 1-26 were originally presented in the application. Claims 2, 11 and 20 have been canceled without prejudice. Claims 1, 3-10, 12-19 and 21-26 stand finally rejected as discussed below. The final rejections of claims 1, 3-10, 12-19 and 21-26 are appealed. The pending claims are shown in the attached Claims Appendix.

Status of Amendments

All claim amendments have been entered by the Examiner. No amendments to the claims were proposed after the final rejection.

Summary of Claimed Subject Matter

One claimed embodiment (see e.g., claims 1, 3-9) provides a method for providing asynchronous network communications between a client and a server. See *Application*, ¶ 13, 14, 34, 87, Figures 14, 15. This embodiment includes configuring a socket for an application on the server. See *Application*, ¶ 88. Once the socket is configured, the claimed embodiment includes issuing a single, continuous mode operation to the socket in response to a user request. The continuous mode operation may be used to reduce redundant accept and receive processing at both the application and operating system levels that occurs using known socket operations. See *Application*, ¶ 87. The single, continuous mode operation may be a single asynchronous accept operation or a single asynchronous receive operation. See *Application*, ¶ 88. In the case of a single asynchronous accept operation, the claimed method includes configuring a listening socket to process a plurality of incoming client connections. See *Application*, ¶ 88, 90, Figure 14, 1408, Figure 15, 1408. In the case of a single asynchronous receive operation, the method includes configuring a client socket to process a plurality of client requests. See *Application*, ¶ 88, 90, 92, Figure 14, 1416, Figure 15, 1416.

Another claimed emboidment includes a computer readable storage medium containing a sockets-based program. See *Application*, ¶ 13, 15, 34, 35, 87. The sockets-based program provides at least a continuous mode accept application programming interface (API) and a continuous mode receive API. See *Application*, ¶ 42, 87-88. In this embodiment, the sockets-based program performs operations for processing messages. The operations may include configuring a listening socket to handle a plurality of incoming client connections, as a result of issuing a single asynchronous accept operation from an application. See *Application*, ¶ 88, 90, Figure 14, 1408, Figure 15, 1408. The operations may also include configuring a client socket to handle a plurality of client requests, as a result of issuing a single, asynchronous receive operation issued by the application. See *Application*, ¶ 88, 90, 92, Figure 14, 1416, Figure 15, 1416.

Still another claimed embodiment includes a system in a distributed computer environment. See *Application*, ¶ 13, 16, 36, 37, 42, 48, Figure 3. The claimed system includes a network facility configured to support a continuous mode network connection between a remote computer and a server computer See *Application*, ¶ 42, 48, 88, Figure 4. The claimed system includes further includes a memory, on the server computer, containing content comprising an application and a plurality of sockets application programming interfaces (APIs), wherein the sockets APIs include at least one of an asynchronous accept operation and an asynchronous receive operation. See *Application*, ¶ 42, 87, 88, Figure 3, 330, 350. The claimed system further includes a processor which, when executing the contents, is configured to perform operations. See *Application*, ¶ 42, Figure 3, 320. In this claimed embodiment, the operations may include in response to a connection request, performing the single asynchronous accept operation to configure a listening socket to receive a plurality of incoming client connections. See *Application*, ¶ 88, 90, Figure 14, 1408, Figure 15, 1408. In this claimed embodiment, the operations may also include, in response to the asynchronous receive request, performing a single asynchronous receive operation to configure a client socket to receive a plurality of client requests. See *Application*, ¶ 88, 90, 92, Figure 14, 1416, Figure 15, 1416.

Grounds of Rejection to be Reviewed on Appeal

1. Claims 1, 4-8, 10, 13-17, 19, 21 and 23-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Admitted Prior Art (APA) in view of *Firth et al.* (US. 5,987,517, hereinafter *Firth*).

2. Claims 3,12 are rejected under 35 U.S.C. 103(a) as being unpatentable over Admitted Prior Art (APA) in view of Firth et al. (US. 5,987,517), as applied to claim 1 above, and further in view of Shah et al (US. Patent 6,175,879 B1).

3. Claims 9, 18, 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Admitted Prior Art (APA) in view of Firth et al (US. 5,987,517), as applied to claim 1 above, and further in view of Joh (US. Patent 6,717,954 B1).

4. Claims 1, 3-10, 12-19, 21-26 are rejected under 35 U.S.C. 102(e) as being anticipated by Bauman (Efficient method for determining record based I/O on top of streaming protocols).

ARGUMENTS

Obviousness of Claims 1, 4-8, 10, 13-17, 19, 21 and 23-26 over Applicants' Admitted Prior Art in view of *Firth* (US. 5,987,517).

The Examiner bears the initial burden of establishing a *prima facie* case of obviousness. *See* MPEP § 2142. To establish a *prima facie* case of obviousness three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one ordinary skill in the art to modify the reference or to combine the reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *See* MPEP § 2143. The present rejection fails to establish at least the first and third criteria.

The Examiner asserts that Applicants' specification (as "admitted prior art" (APA)) discloses the continuous mode operations recited by claims 1, 10, and 19. For example, claim 1 recites "in response to a request from the client, issuing a single, continuous mode operation to the socket." Claims 10 and 19 each recite a corresponding limitation. Although the Examiner argues that the APA discloses this limitation "at p. 3, lns. 13-16 and p.3 lns. 9-10," these passages are in fact directed to prior art methods that require multiple accept() or receive() operations be performed as part of a socket-based communication exchange. Set out in full, this passage provides:

> Sockets accept connections and receive data from clients using well-known "accept" and "receive" semantics, respectively. The accept and receive semantics are illustrated in FIGS. 1 and 2 as accept ( ) and asyncAccept ( ), respectively, and as receive ( ) and asyncReceive ( ), respectively. Sockets accept/receive semantics are either synchronous (FIG. 1) or asynchronous (FIG. 2). Synchronous accept/receive APIs accept connections and receive data in the execution context issuing the API.

Application, p.3, 9-15. Nothing in this passage (or the APA generally) discloses a method of socket-based communication that includes issuing a "continuous mode operation to the socket." Rather, the passage cited by the Examiner describes the

existence of accept() and receive() calls as known socket-based communication semantics, a point Applicants do not dispute. As Applicants point out in the APA, however, the prior art methods of actually <u>using</u> socket-based accept/receive communications often require the inefficient use of <u>multiple</u>, accept() and receive() calls:

> As suggested by the loops shown in FIG. 1 and FIG. 2, accept and receive processing for both synchronous and asynchronous environments is highly repetitive with little variance in the parameters. As a result, a server may service thousands of accepts and receives in a short period of time. Because this is such a common path, it would be desirable to eliminate or reduce redundant accept and receive processing.

Application, ¶ 11. Distinctions between the Admitted prior art and the "continuous mode operation" of the present claims are further detailed in Applicants' specification:

> Only a single asynchronous continuous receive operation 1416 is issued for each client connection and results in a pending receive data structure (not shown) being placed on the pending queue 1410. A loop 1417 defines repetitious request processing performed by the main thread 1402. <u>Note that the loop 1417 does not include redundant accept operations.</u> Once a completed client record has been received, a completed receive data structure (not shown) is placed on a receive completion queue 1420. Completed receives are dequeued from the completion queue 1420 by an asynchronous wait operation 1422 issued by a worker thread 1404A. A loop 1424 defines repetitious request processing performed by the worker thread 1404A. <u>Note that the loop 1424 does not include redundant receive operations.</u>
>
> Accordingly, as is evident by comparison of FIG. 14 with FIGS. 1 and 2, various redundant processing has been eliminated. Comparing FIG. 14 to FIG. 2, for example, the asynchronous accept operation 208 has been taken out of the loop 215 and replaced with the asynchronous continuous accept operation 1408. Further, the loop 224 has been eliminated by virtue of utilizing the record definitions 364/366 and the need for redundant asynchronous receives 222 issued by a worker thread has been eliminated.

Application, ¶ 87, 88.

Moreover, the Examiner concedes that the "APA does not explicit teach [sic] the single asynchronous operation." *See Final Office Action*, p. 3. That is, the examiner

appears to recognize that the prior art socket-based communication discussed in the APA requires the use of multiple accept() / receive() calls.

Nevertheless, the Examiner argues that this APA in view of *Firth* renders the techniques for socket-based communications recited by the present claims obvious. However, the material cited from *Firth* (and *Firth* generally) fails to disclose techniques for managing socket-based communications. Rather, *Firth* is directed to an API of functions that provide an abstraction of data-communication techniques. The abstraction allows developers to create high-level applications without having to understand or manage the details of an underlying data communication mechanism. That is, the abstraction provided by the "Internet API" allows developers to create software applications without having to understand how a particular socket-based communication may occur. For example, *Firth* provides;

> In the preferred embodiment of the present invention, calls to two of the reentrant Internet API functions (e.g., InternetOpen(. . .),InternetConnect(. . . ,FTP, . . .), which will be explained in detail below) will initialized an Internet session, establish a connection, and manage all the underlying details including the FTP protocol, the communication facilities required (e.g., a socket connection), ... <u>The Internet application program does not have to include source code to establish an Internet connection, handle the FTP protocol, the communications facilities, or the underlying protocols. All of these details are abstracted in the Internet API and are hidden from or transparent to the application program.</u>

Firth, 3:37-52. As the emphasized passage makes clear, *Firth* discloses techniques that allow an application developer to compose an application without an understanding of "the underlying protocols." Not surprisingly, therefore, *Firth* fails to teach or suggest mechanisms for providing asynchronous network communications, in the manner claimed.

In asserting that "*Firth* teaches single asynchronous operation," Final office Action, p. 3, the Examiner cites three passages directed to aspects of the "Internet API" disclosed in Firth. These passages highlight that *Firth* is directed to functions provided by an "Internet API," and not to issuing a single, continuous mode operation selected

from a single asynchronous accept operation and a single asynchronous receive operation, as recited by claims 1, 10, and 19.

First the Examiner cites cited *Firth*, 16:23-27, which provides:

> As was just described, a single call to the InternetOpen ( ) function from the internet API provides a client application with the ability to select the type of Internet access, select a proxy for a first level of security, ... [or perform other various actions]."

As disclosed in *Firth*, the InternetOpen ( ) function is used to '"initialize the application's use of the reentrant Internet API functions. *Firth*, 9:35-38. In other words, an application calls the InternetOpen function in order to use other functions provided by the "Internet API" disclosed by *Firth*. For example, *Firth* describes how the InternetOpen () function must be called prior to a call to another "Internet API," function, InternetConnect():

> As an example, InternetConnect(), which is a first level dependent function, cannot be called until InternetOpen () (an independent function) is first called and returns a valid Internet handle, which is a required argument for InternetConnect() call. If an application desires to find the first file located during an FTP session with a connection to the Internet, InternetOpen () is called and the Internet handle that is returned is used as an argument for a call to InternetConnect(. . . ,FTP, . . .) to establish an FTP application protocol session. Finally, the Internet handle returned from InternetConnect() is used as an argument in a call to FtpFindFirstFile(). Other dependent functions use handles from the next higher level in a similar manner.

Firth, 11:48-61. Applicants submit that initializing the "Internet API" of *Firth* using the InternetOpen() function of the "Internet API" fails to disclose "issuing a single, continuous mode operation" to a socket using a "single asynchronous accept operation" or a "single asynchronous receive operation." Rather, the IntenetOpen() function allows an application developer to make function calls to other functions of the "Internet API" disclosed in *Firth*. These functions operate independently from the mechanism used to manage socket-based communications. In fact, this is the whole point of the "Internet API": to provide an API where the details of a socket-based communication are "abstracted in the Internet API and are hidden from or transparent to the application program." *Firth*, 3:50-52.

The other two passages from *Firth* cited by the Examiner provide:

> The Internet API provides a function call to set up call back routines (see, e.g., InternetSetStatusCallback() in Appendix A) for asynchronous function operation. A single operating system thread is used to handle all asynchronous Internet API function operations. However, multiple operating system threads could also be used.
>
> *Firth*, 15:42-46.
>
> [Using the"InternetConnect()" function call] an application can communicate common information about several requests using a single function call. In addition, this single application protocol connection function call provides flexibility for adding new or additional Internet application protocols.

Firth, 18:15-20. The first of these two passages describes a function of the "Internet API" used to set up call back routines managed by a single operating system thread (although "multiple operating system threads could also be used"). A callback function is a set of executable code that is passed as a parameter to other code. The passage appears to disclose that the "internet API" may include a callback function processed by ether a single, or multiple, operating system threads.

The second of these two passages describes the use of an "InternetConnect()" function call provided by the "Internet API." Specifically, this passage describes how the "InternetConnect()" function call may be used to initiate a connection using many different communication protocols, as opposed to having multiple function calls, one for each different protocol. Applicants submit that the Examiner is confusing the recited claim limitations of using a single asynchronous accept operation to configure a listening socket to process a plurality of incoming client connections (or the limitation of using a single asynchronous receive operation to configure a client socket to process a plurality of client requests) with a description of a single function call that may take varying parameters. Moreover, each of these passages describes how different functions of Firth's "Internet API" may be used to create a "high-level" application where the socket-based communications are "abstracted in the Internet API and are hidden from or transparent to the application program." *Firth*, 3:50-52.

What none of these three passages, either alone or in combination, describes, however, is using a single asynchronous accept operation to configure a listening socket to process a plurality of incoming client connections or using a single asynchronous receive operation to configure a client socket to process a plurality of client requests. This is wholly unsurprising as *Firth* describes an Internet API that is provided, in part, to conceal just these kinds of details from an application programmer. See, e.g., *Firth*, 3:37-52.

Finally, the Examiner asserts that "it would have been obvious to one of the ordinary skill in the art at the time the invention was made to combine the teaching of APA and *Firth* because *Firth's* single asynchronous operation would improve flexibility of APA's system by adding new or additional Internet application protocols for establishing communications with a variety of computer networks." *See Final Office Action*, p. 3. It appears that the Examiner is paraphrasing one of the passages cited in the rejection of claims 1, 10, and 19. Specifically, the Examiner paraphrases the third passage from *Firth* regarding the InternetConnect() function call. *Firth* goes on to provide: "However, the [InternetConnect()] function call and the number off arguments would remain the same and provide a consistent interface for applications." *Firth*, 18:23-26. Applicants submit that the "InternetAPIs" use of a single overloaded "InternetConnect()" function calls to establish multiple connections for different communication protocols is unrelated to use of a "a single asynchronous accept operation" and "a single asynchronous receive operation" to reduce the number of accept() or receive() calls made as part of a socket-based communication exchange, as recited by claims 1, 10, and 19.

For all the foregoing reasons, Applicants submit that independent claims 1, 10, and 19 are allowable. Applicants respectfully request, therefore, withdrawal of this rejection and the allowance of claims 1, 4-8, 10, 13-17, 19, 21 and 23-26.

Obviousness of claims 3 and 12 over APA in view of *Firth* and further in view of *Shah*

Claims 3 and 12 claims depend from claims 1 and 10, respectively. As Applicants believe the above remarks demonstrate that APA in view of *Firth* fails to

render the independent claims obvious, Applicants believe that these dependent claims are allowable over APA in view of *Firth* and further in view of *Shah*.

Obviousness of claims 9, 18, and 22 over APA in view of *Firth* and further in view *Joh*

Claims 9, 18, and 22 claims depend from claims 1, 10, and 19, respectively. As Applicants believe the above remarks demonstrate that APA in view of *Firth* fails to render the independent claims obvious, Applicants believe that these dependent claims are allowable over APA in view of *Firth* and further in view of *Joh*.

Anticipation of claims 1, 3-10, 12-19, 21-26 by related application *Bauman*

Claims 1, 3-10, 12-19 and 21-26 are rejected under 35 U.S.C. § 102(e) as being anticipated by *Bauman et al.* (U.S. 2003/0097488, hereinafter *Bauman*).

During an informal telephone conference on February 21, Jon K. Stewart, attorney for Applicant, and Examiner Troung discussed the availability of *Bauman* as a reference against the present Application. On the basis of this telephone conference, Applicants' believe the Examiner agrees that the *Bauman* reference is not prior art under 35 § U.S.C. 102(e) against the present application.

More specifically, *Bauman* fails to qualify as a reference under 35 § U.S.C. 102(e) because both share the same priority date. The present application was filed on January 4, 2002 with a preliminary amendment. The preliminary amendment includes a priority claim to U.S. 09/990,850 with a U.S. filing date of November 21, 2001. In point of fact, the present rejection claims priority to the *Bauman* reference. The filing receipt received by Applicants reflects the priority claim to *Bauman*. Thus, the present application and *Bauman* share the same date, for purposes of determining the availability of a prior art reference under 35 U.S.C. § 102. Therefore, Applicants respectfully request that the rejection be withdrawn.

RECEIVED
CENTRAL FAX CENTER
JUL 28 2006

CONCLUSION

The Examiner errs in finding:

- that claims 1, 4-8, 10, 13-17, 19, 21 and 23-26 are obvious over APA in view of *Firth*.
- that claims 3 and 12 are obvious over APA in view of *Firth* and further in view of *Shah*
- that claims 9, 18, and 22 are obvious over APA in view of *Firth* and further in view *Joh* (US. Patent 6,717,954 B1)
- that claims 1, 3-10, 12-19, 21-26 are anticipated by related application *Bauman* (Efficient method for determining record based I/O on top of streaming protocols).

Respectfully submitted, and
S-signed pursuant to 37 CFR 1.4,

/Gero G. McClellan, Reg. No. 44,227/
Gero G. McClellan
Registration No. 44,227
Patterson & Sheridan, L.L.P.
3040 Post Oak Blvd. Suite 1500
Houston, TX 77056
Telephone: (713) 623-4844
Facsimile: (713) 623-4846
Attorney for Appellant(s)

CLAIMS APPENDIX

1. (Previously Presented) A computer-implemented method ~~of~~ for providing asynchronous network communications between a client and a server, comprising:

configuring a socket for an application on the server;

in response to a request from the client, issuing a single, continuous mode operation to the socket, wherein the single, continuous mode operation is selected from at least one of:

a single asynchronous accept operation, configuring a listening socket to process a plurality of incoming client connections; and

a single asynchronous receive operation, configuring a client socket to process a plurality of client requests.

2. (Canceled)

3. (Previously Presented) The computer-implemented method of claim 1, further comprising, configuring the client socket, with the single asynchronous receive operation, to recognize a format of each of the plurality of client requests, whereby the client socket is configured to receive the client requests without invoking the application until the request is completely received.

4. (Previously Presented) The computer-implemented method of claim 1, wherein the single, continuous mode operations are issued from a main thread of the application.

5. (Previously Presented) The computer-implemented method of claim 1, wherein issuing the single asynchronous receive operation comprises:

placing a single pending receive data structure on a pending queue;

for each completed client request, copying contents of the pending receive data structure to a completed receive data structure queued on a receive completion queue.

6. (Previously Presented) The computer-implemented method of claim 1, wherein issuing the single asynchronous accept operation comprises:

placing a single pending accept data structure on a pending queue;

for each of the plurality of incoming client connections, copying contents of the single pending accept data structure to a completed accept data structure queued on a accept completion queue, wherein the single pending accept data structure remains on the pending queue.

7. (Previously Presented) The computer-implemented method of claim 6, wherein issuing the single asynchronous receive operation comprises:

placing a single pending receive data structure on a pending queue;

for each completed client request, copying contents of the pending receive data structure to a completed receive data structure queued on a receive completion queue.

8. (Previously Presented) The computer-implemented method of claim 1, further comprising, for each completed client request, acquiring a buffer from system supply memory to contain the completed client request.

9. (Previously Presented) The computer-implemented method of claim 8, wherein allocating the buffer comprises sizing the buffer according to a size of the completed client request.

10. (Previously Presented) A computer readable storage medium containing a sockets-based program comprising at least one of a continuous mode accept application programming interface and a continuous mode receive application programming interface, wherein the sockets-based program, when executed, performs operations for processing messages, the operations comprising at least one of:

configuring a listening socket to handle a plurality of incoming client connections, as a result of issuing a single asynchronous accept operation from an application; and

configuring a client socket to handle a plurality of client requests, as a result of issuing a single, asynchronous receive operation issued by the application.

11. (Canceled)

12. (Previously Presented) The computer readable medium of claim 10, further comprising, configuring the client socket, with the single asynchronous receive operation, to recognize a format of each of the plurality of client requests, whereby the client socket is configured to handle receiving the client requests without invoking the application until the message is completely received.

13. (Previously Presented) The computer readable medium of claim 10, wherein the single asynchronous accept operation and the single asynchronous receive operation are issued from a main thread of the application.

14. (Previously Presented) The computer readable medium of claim 10, further comprising, when the single asynchronous receive operation is issued:

placing a single pending receive data structure on a pending queue;

for each completed client request, copying contents of the pending receive data structure to a completed receive data structure queued on a receive completion queue.

15. (Previously Presented) The computer readable medium of claim 10, further comprising, when the single asynchronous accept operation is issued:

placing a single pending accept data structure on a pending queue;

for each of the plurality of incoming client connections, copying contents of the single pending accept data structure to a completed accept data structure queued on a accept completion queue, wherein the single pending accept data structure remains on the pending queue.

16. (Previously Presented) The computer readable medium of claim 15, further comprising, when the single asynchronous receive operation is issued:

placing a single pending receive data structure on a pending queue;

for each completed client request, copying contents of the pending receive data structure to a completed receive data structure queued on a receive completion queue.

17. (Original) The computer readable medium of claim 10, further comprising, for each completed client request, acquiring a buffer from system owned memory space to contain the completed client request.

18. (Original) The computer readable medium of claim 17, wherein allocating the buffer comprises sizing the buffer according to a size of the completed client request.

19. (Previously Presented) A system in a distributed computer environment, comprising:

a network facility configured to support a continuous mode network connection between a remote computer and a server computer;

a memory, on the server computer, containing content comprising an application and a plurality of sockets application programming interfaces (APIs), wherein the sockets APIs comprise at least one of an asynchronous accept operation and an asynchronous receive operation;

a processor which, when executing the contents, is configured to perform operations comprising at least one of:

in response to a connection request, performing the single asynchronous accept operation to configure a listening socket to receive a plurality of incoming client connections; and

in response to the asynchronous receive request, performing a single asynchronous receive operation to configure a client socket to receive a plurality of client requests.

20. (Canceled)

21. (Original) The system of claim 19, wherein the content of the memory further comprises a system owned memory space and wherein the operations further comprise:

for each completed client request, acquiring a buffer from the system owned memory space to contain the completed client request.

22. (Original) The system of claim 19, wherein the content of the memory further comprises a system owned memory space and wherein the operations further comprise:

for each completed client request, acquiring a buffer from the system owned memory space to contain the completed client request, wherein the buffer is sized according to a size of the completed client request.

23. (Previously Presented) The system of claim 19, wherein the content of the memory further comprises a pending queue on which a single pending accept data structure is queued as a result of the single asynchronous accept operation.

24. (Original) The system of claim 23, wherein the content of the memory further comprises an accept completion queue to which contents of the pending accept data structure are copied upon receiving a client connection on the listening socket and wherein the pending accept data structure remains on the pending queue.

25. (Previously Presented) The system of claim 19, wherein the content of the memory further comprises a pending queue on which a single pending receive data structure is queued as a result of the single asynchronous receive operation.

26. (Original) The system of claim 25, wherein the content of the memory further comprises a receive completion queue to which contents of the pending receive data structure are copied upon receiving a completed client request on the client socket and wherein the pending receive data structure remains on the pending queue.

EVIDENCE APPENDIX

None.

RELATED PROCEEDINGS APPENDIX

None.